Patent
237/116
OI7011472001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re the Application of:	Group Art Unit: 2172
Sashikanth Chandrasekaran, et al.	Examiner: To, Baoquoc N.
Serial No.: 09/265,489	
Filed: March 9, 1999	
For: METHOD AND SYSTEM FOR RELIABLE ACCESS OF MESSAGES BY MULTIPLE CONSUMERS	

DECLARATION OF SASHIKANTH CHANDRASEKARAN UNDER 37 C.F.R. § 1.131

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

I, Sashikanth Chandrasekaran, hereby declare as follows:

1. I am co-inventor of the invention described in the above application.

2. I am employed as a software and technology developer at Oracle Corporation at its offices in Redwood Shores, California.

3. As evidenced by the document attached to this affidavit as Exhibit A, prior to January 15, 1999, my co-inventors and I had conceived and diligently reduced to practice the subject matter of the above application. Exhibit A is a portion of a Design Specification dated prior to January 15, 1999 which describes the design specification for implementing database tables and related structures for managing message data to be accessed by multiple recipients.

4. Section 2.2.4 of Exhibit A describes history management processes that are implemented in a software program that was created and reduced to practice prior to January 15, 1999 which embodied the subject matter of the above application.

5. Section 3.4.1 of Exhibit A describes database fields that are employed to manage history information for the processing of messages, as well as procedures for proving to users and updating such information. Specific information to track for history records are also provided in this section. This database structure was implemented in a software program that was created and reduced to practice prior to January 15, 1999 which embodied the subject matter of the above application.

6. Section 3.4.2 of Exhibit A describes an algorithm to update and manage index records relating to the messages and message recipients. This process was implemented in the software program that was created and reduced to practice prior to January 15, 1999 which embodied the subject matter of the above application.

7. I further declare that all statements made herein of my own knowledge are true and all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Date: Aug 18 2005

[illegible]

Sashikanth Chandrasekaran

Design Specification for AQ Propagation , RDBMS, 8.1

Project ID: aq_propagation

Version: [redacted]

Status: Approved

Author: Sashi Chandrasekaran, Ashok Saxena

[redacted]

Version	Reviewers	Changes
[redacted]	arsaxena	Creation
[redacted]	[redacted]	[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

2.2.4 History Management

History management for multi-consumer queues leaves a lot to be desired. There are two fundamental problems to history management: storage and reference counting.

- Storage

  The history information is stored as a varray object collection. The current interface to varray collections retrieves and unpickles the entire collection. AQ uses C interfaces to navigate through the collection and update the history element for the appropriate consumer. The updated collection is written entirely into the database for history tracking.

- Reference Counting

In Oracle 8.0.4 a reference count is maintained as a separate column with each message. Dequeuers decrement the reference count and the last dequeuer (that finds the reference count to be zero) deletes the message from the queue table. Needless to say when several consumers are trying to dequeue the same messages a convoy will quickly form behind the reference count hotspot. We intend to eliminate the hot spot by de-coupling the reference counting from the dequeuers by entrusting the queue monitor with the task of performing the garbage collection (i.e delete messages that have been dequeued by all consumers). It already performs the task of deleting messages that have expired.

We propose to keep the history information in a separate IOT. This will reduce the contention among the multiple consumers to update the history and eliminate the need to lock the queue table entry for the message. The time manager index will be enhanced and the time manager's responsibilities will be increased to update the state of the message to "PROCESSED". If the queue has a non-infinite retention time, the time-manager will not update the state of the message to processed. Instead, the time-manager will only remove the message when the retention time is complete. This is not a problem because the state of the message can de deduced from the history entries in the history IOT. When a message is dequeued by a consumer, its entry in the message table is updated to the new removal time (current time + retention). When the time manager encounters this entry it will check the history and if all recipients have processed the message it will remove the message from the queue table and also the history rows. Since the history IOT is indexed on message id as the leading primary key the time manager can efficiently determine the status of a message.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

3.4 History Management

3.4.1 Data Structures

When a queue table is created, three additional IOTs are created to store the message meta data. The message data and message properties are kept in the queue table. The queue table in 8.1 will be modified for changes in message properties, but they are not relevant to the history management. The three additional IOTs are:

- A dequeue index to maintain the sort-order of messages for each recipient. This index will have the same structure as in Oracle 8.0.
- A history index that maintains the history of processing of every message. The columns in this IOT are as follows:

  a. msgid - unique identifier of the message

  b. rowid - location of the message in the queue table.

  c. address - address of the recipient.

  It is the source queue name (without the schema name appended to it) if the consumer will dequeue messages directly from the source queue. The address supported by AQ propagation will be of the form [schema.]queue[@database_link]. Messages are propagated to the destination queue specified by the address. AQ does not require global names be set to TRUE, however it is recommended. The database link name is resolved in the context of the owner of the source queue.

  d. protocol - protocol field of the recipient structure.

  This field qualifies the address. It is the session-level protocol (e.g. dblink/TIB) used to propagate messages to the destination queue. It is 0 if the address is a database link address or if the consumer dequeues the message from the local queue.

  e. consumer_name - name of the agent (recipient) that dequeued the message.

  f. txn_id - transaction id of the dequeuing transaction.

  g. deq_time - time of dequeue.

  h. deq_user - database schema id of dequeuer.

  j. propagated_msgid - message id of the enqueued message in the destination queue.

  This is NULL if the address is NULL.

  k. retry_count - # times message was dequeued in remove mode (and aborted).

  Columns a, c, d and e form the primary key. We may choose to include the other columns also as part of the primary key to simplify access to these columns (Key columns are easier to extract than non-key columns and also do not have the complexity of an overflow segment). Key-compression will not be used since we do not expect the prefix (msgid) to be repeated often.

- A time-manager index that maintains the list of time-management activities. The time-manager index has four columns:

  a. time - absolute time at which time-manager has to perform an operation.

b. msgid - message id of message that needs to be acted upon.

c. action - a description of the action that needs to be performed. The possible values are:

1. MAKE_READY - make message available for dequeue to consumers after the delay time has passed.

2. EXPIRE - move message to exception queue if message has not yet been processed.

3. REMOVE - remove message after the retention time has passed.

d. transaction_id. This is the transaction_id of the transaction that inserts the time-management entry. This is needed to generate a unique key, since two consumers can dequeue the same message and post the time-manager to perform an action at the same time. This is set if the action is REMOVE.

Columns a, b and d form the primary key. This IOT is similar to the time-manager index for Oracle 8.0 queue tables. The differences are:

a. The IOT stores the msgid of the message rather than its rowid.

b. There is an action column to help the time-manager determine what time-management activity needs to be performed on the message. In theory, this column is superfluous because the time-manager can deduce what action needs to be performed based on the history information in the historytable.Oracle8.0'stime-managerindexdeducedwhataction needs to be performed based on the state of the message in the queue table.

c. There could be multiple rows for the same message in the index. In fact, there could be up to one row for each agent that dequeues the message from the queue table. This is because each agent that dequeues messages independently notifies the time-manager without knowledge of the state of the message with respect to other recipients.

3.4.2 Design Description

We illustrate the use of these index structures using a simple example. Let us assume that a queue table, say qt, has been created. Call the dequeue sort order index qt_i, the history index as qt_h and the time-manager index as qt_t. Let us say a message is enqueued in queue q with the following properties: messageid = m, delay = d, expiration = e, retention time = r, recipients = {r1, r2@boston} where r1 is a local consumer and boston is a remote database. The acknowledgment mode for this message is assumed to be ACK_DEQUEUED (a propagator and dequeuer perform similar actions if the acknowledgment mode is ACK_PROPAGATED or NO_ACK).

When the message is enqueued at rowid = rid, the index structures are updated as follows:

1. insert one key into qt_i for the propagator. This step is identical to Oracle 8.0. This step is necessary so that the propagator can dequeue the message without waiting for the delay time.

2.Ifdisnon-NULLinsertkeyintoqt_twithvalue[d,m,MAKE_READY,txnid]elseifeisnon-NULL insert key into qt_t with value [e, m, EXPIRE, txnid]

3. array insert two keys into qt_h with values [m, rid, r2@boston, 0, r2, NULL, NULL, NULL, NULL, 0] and [m, rid, q,0, r1, NULL, NULL, NULL, NULL, 0]. This step will substitute generating the history collection in an 8.0 queue table.

When the delay time has passed the time-manager performs the following actions.

1. for each entry in qt_h where msgid = m and address = q and txn_id = txnid insert key into qt_i to enable consumer to dequeue message.

2. update the qt_t key to [e, m, EXPIRE, cur_txnid] if e is non-NULL.

Agent r1 performs the following steps after dequeuing message m.

1. Delete its index entry from qt_i.

2. Update deq_time, deq_user, txn_id columns in qt_h for row with consumer_name = r1.
3. If retention_time is not NULL, insert key [r, m, REMOVE, cur_txnid] into qt_t else if queue has no retention, insert key [gettimeofday(),m, REMOVE, cur_txnid].

The propagator updates the propagated_msgid column in qt_h and deletes the index entry from qt_i as soon as m is successfully propagated to boston. The deq_time column in qt_h and time_manager index qt_t are updated only on receipt of acknowledgment from boston that r2 has processed the message.

The time-manager marks the message as expired at time e, if either r1's or r2's deq_time columns in qt_h is NULL. Likewise, it removes the message m at time r only if r1 and r2 deq_time columns are non-NULL. In all cases the time-manager removes the index entry from the time-manager-index when it processes the entry, regardless of whether processing it resulted in any state change or not. When a message expires, the history keys in qt_h are copied over to a different queue table if the exception queue resides in a different queue table. The history keys are deleted along with the message itself when the application uses dequeue-by-message-id to remove the message from the exception queue.

[redacted]

[redacted]

[redacted]